Quality Attributes for Service-Oriented Architectures

Liam O'Brien
Lero,
University of Limerick,
Ireland.
liam.obrien [at] lero.ie

Paulo Merson
Software Engineering
Institute
Pittsburgh, PA, USA.
pfm [at] sei.cmu.edu

Len Bass
Software Engineering
Institute
Pittsburgh, PA, USA.
ljb [at] sei.cmu.edu

Abstract

The SOA approach is a very popular choice today for the implementation of distributed systems. The use of SOA or more specifically the Web services technology is an important architecture decision. An architect should understand how different quality attributes for a system are impacted by that decision. While there are significant benefits with respect to interoperability and modifiability, other qualities such as performance, security and testability are concerns. This paper discusses how the different quality attributes of a system can be positively or negatively affected by the use of such technology. It describes the factors related to each attribute, as well as possible tradeoffs and existing efforts to achieve that quality. The paper also discusses open issues in service level agreements that are used to contract the level of service quality between service providers and users.

1. Introduction

A service-oriented architecture (SOA) is an architectural approach for building systems where there are components that are service users and/or service providers. But what essentially characterizes an SOA is the service. In this context, a service is a distributed component with the following characteristics: is self-contained; has a published interface that abstracts the underlying logic; is location transparent; can be implemented in different languages or platforms and still interoperate; is discoverable and dynamically bound. The most prominent technology that implements the SOA architectural approach today is Web services. Other SOA technologies include Common Object Request Broker Architecture (CORBA) and the Jini network technology (http://www.sun.com/jini).

Software architecture is the bridge between business goals and the software system. Choosing and designing an architecture that satisfies the functional as well as the quality attribute requirements (reliability, security, performance, etc.) is vital to the success of the system. The quality attribute requirements in particular drive the software architecture design [1].

An architect considering using the general SOA approach or the Web services technology in particular should understand how different quality attributes will be positively or negatively impacted, and what the tradeoffs involved are. Though some work has been done on how SOA affects particular qualities such as security and interoperability, a more thorough examination of the relationship between SOA and quality attributes is needed. Such an examination is the objective of this paper.

In some situations, service level agreements (SLAs) are used by service providers and users to contract a particular level of service quality. These SLAs have to be defined, monitored and enforced so that service functionality and data can be predictably and contractually delivered between the providers and users. For both providers and users it is important to understand and map business drivers to quality attribute requirements and to clearly articulate these in the SLAs. Equally important is to have processes to monitor the quality of the service provided and to define policies that deal with situations where the contracted level of service is not met.

The paper presents in sections 2 to 10 a discussion of SOA aspects related to ten different quality attributes. Section 11 outlines open issues in service level agreements (SLAs) and the handling of quality attributes in SOA lifecycle management. Section 12 ends the paper with concluding remarks.

2. Interoperability

Interoperability refers to the ability of a collection of communicating entities to share specific information and operate on it according to an agreed-upon operational semantics [2]. Increased interoperability is the most prominent benefit of SOA, especially when we consider Web services technology.

Today, mainstream development platforms—such as Microsoft's .NET and Sun's Java Platform, Enterprise Edition (Java EE), as well as open source alternatives (e.g., Perl and PHP)—provide frameworks to implement Web services. Service users and providers implemented in disparate platforms using different languages can

International Workshop on Systems Development in SOA Environments (SDSOA'07)
0-7695-2960-7/07 $20.00 © 2007 IEEE

interact transparently through a call-and-return mechanism. That is possible because Web services define the interface format and communication protocols but do not restrict the implementation language or platform. Other SOA technologies—such as Jini and CORBA—have not yet produced enough of a following to become a universal standard for interoperability [3].

However, the Web services goal of cross-vendor and cross-platform interoperability begins to fall short when services start to use features beyond the two basic standards: Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP). Over the last few years, many Web services standards (e.g., Business Process Execution Language [BPEL], WS-Security, WS-ReliableMessaging and ebXML) have emerged from a number of standards bodies. Web services development platforms do not implement the same versions of all standards, so interoperability may not be as seamless in practice as it is in theory. Furthermore interoperability using Web services is purely syntactic unless the WSDL document describing the service contains enough information to determine the correct meaning of the exchanged data (in practice this is hard to find).

To promote the interoperability of Web services across platforms, applications, and programming languages, the Web Services-Interoperability Organization (WS-I) was chartered in 2002. WS-I publishes *profiles* that prescribe adherence to a group of specific versions of well-defined standards. It is also their goal to provide tools to certify conformance with the profiles. Many Web services products were updated in recent years because of this initiative; it is not uncommon now to find "compliant with WS-I Basic Profile 1.1" in data sheets. WS-I has created a few profiles and other deliverables but still has a lot of work to do to cover all layers and standards in the Web services stack.

Open issues: Semantic interoperability is an area where work is being done but further research is needed to achieve seamless interoperability between systems. Also work needs to be done on developing mechanisms for evaluating compliance with the WS-I profiles.

3. Performance

Performance can mean different things in different contexts. In general, it is related to response time (how long does it take to process a request), throughput (how many requests overall can be processed per unit of time), or timeliness (ability to meet deadlines, i.e., to process a request in a deterministic and acceptable amount of time). In most cases, performance is negatively affected in SOAs. The architecture should be carefully designed and evaluated prior to implementation to avoid performance pitfalls. The key factors in SOA that contribute to performance issues are:

- SOA involves distributed computing. Service providers and service users are normally located on different machines. The need to communicate over the network increases the response time. In addition, the network usually does not guarantee deterministic latency, making SOA a poor choice, for example, for real-time systems, where timeliness is a strict requirement.
- The interaction protocol sometimes requires a call to a directory of services to locate the desired service. This extra call increases the total time needed to perform the transaction. One way to reduce the response time and improve throughput is to prevent the call to the directory by having the location of the provider end point hard-coded (or cached after the first lookup) in the service user. However, hard-coding reduces flexibility, and caching must be reestablished after failure of a service provider instance when another replica is found.
- The ability to make services on different platforms interoperate seamlessly has a performance cost. Intermediaries are needed to perform data marshalling and handle all communication between service user and provider. Depending on the SOA technology or framework being used, stubs, skeletons, SOAP engines, proxies, ESBs, and other kinds of elements are in place. All such intermediaries cause some performance overhead.
- The use of a standard messaging format increases the time needed to process a request. For example, in the Web Services technology, the use of XML has a great impact on performance. XML is text-based and messages can be 10 to 20 times larger than the equivalent binary representation, so transmitting them over a network takes longer. Moreover, XML messages have to be processed before any operation is performed. XML processing consists of at least three distinct activities, all of which are CPU and memory intensive: parsing, validation and transformation.

On the positive side, SOA provides location transparency. Service users do not necessarily know the location of the service until they look it up in the registry. Thus, a deployed service can be moved from location to location without affecting the users. This feature permits the deployment of services to multiple locations, which can be allied to a load-balancing strategy to improve the total throughput and availability of a system.

Open issues: work is being done on building performance models for component-based and SOA systems but further research needs to be done on building models of the highly complex run-time environment for SOA-based systems and deriving the performance parameters for these models.

4. Security

Although security denotes different things with respect to software systems, in general, it is associated with four principles: *(a)* confidentiality, which ensures that access to information/services is granted only to authorized subjects; *(b)* authenticity, which is related to trust that the indicated author/sender is the one responsible for the information; *(c)* integrity, which guarantees that information is not corrupted; and *(d)* availability, which ensures that the service is available in a timely manner.

Security is a major concern for SOA and Web services. Architects should pay heed to some characteristics that are inherent to SOAs and directly impact security:

- Messages may be in text format and contain metadata. Therefore, someone intercepting a message may clearly see a 16-digit number as well as metadata revealing that the number is the value of a credit card field. Encryption should be in place to preserve confidentiality but encryption has the effect of increasing the message size.
- A system built using an SOA approach may encompass services provided by third-party organizations. The identity of the external service provider should be authenticated, but sometimes authentication is not enough. For instance, if the system sends classified data to the external service, the data should be protected not only when it is transmitted but also when it is stored.
- Service providers may need to enforce access restrictions based on the identity of the user. In that case, an authorization mechanism should be in place. Single sign-on, which is common in SOA-based systems that require access to services on multiple machines, may affect other attributes due to the session information that has to pass from request to request.
- An SOA solution may rely on looking up services in a public directory. It is important to ensure that information in the directory is up to date and was added by valid publishers.

Web services solutions have been addressing some of the security concerns at the network infrastructure level. For example, Web servers that host Web services can be configured to use Secure Sockets Layer (SSL) and digital certificates to encrypt data transmission and authenticate the communicating parties. In intranet solutions, Kerberos is an option—users receive a ticket for access to each Web service they have permission to use. However, these solutions merely help to protect point-to-point interaction: A comprehensive mechanism that covers end-to-end security is required.

In 2002, IBM, Microsoft, and VeriSign proposed Web Services Security as a comprehensive security model for Web services. Besides the core WS-Security policy for message protection, the original proposal contained a roadmap of complementary security specifications [4]. These specifications (WS-Authorization, WS-Privacy, WS-Trust, WS-Federation, WS-Policy, and WS-SecureConversation) are gradually developing into standards. The WS-Security specification was submitted to OASIS, and the first version was approved in 2004 [5]. WS-Security defines a standard set of SOAP extensions that can be used to provide message content integrity and confidentiality. It accommodates a variety of security models and encryption technologies and is extensible to support multiple security token formats.

Two other proposed standards relevant to Web services security are Security Assertions Markup Language (SAML) and eXtensible Access Control Markup Language (XACML). SAML provides a standard, XML-based format to exchange security information between different security agents over the Internet. It allows services to exchange authentication, authorization, and attribute information without organizations and their partners having to modify their current security solutions [3]. XACML complements SAML by providing a language to specify role-based, access control rules in a declarative format.

Security mechanisms may have a negative impact on performance, modifiability and interoperability. Adherence to security standards is important to preserve interoperability. WS-I has been working on a basic security profile that will ensure interoperability of security features among compliant vendors [6].

The architect should also look into the network configuration required by the chosen SOA technology. For example, if a service user interacts with a remote provider via the Internet using CORBA, then the firewall on both ends probably needs to permit Internet Inter-ORB Protocol (IIOP) communication. On the other hand, in a Web services solution, firewall rules don't need to change because the SOAP interaction is over a protocol that is normally open (e.g., HTTP or SMTP).

Open issues: though there are many emerging standards for security, how these are actually applied in the development of an SOA-based system and the effect that the use of particular security mechanisms have on other attributes still needs further research.

5. Reliability

Reliability is the ability of a system to keep operating over time without failure [8]. Several aspects of reliability are important within an SOA, particularly the reliability of the messages that are exchanged between service users and providers, and the reliability of the services themselves.

5.1. Message Reliability

Services are often made available over a network with possibly unreliable communication channels. Connections break and messages fail to get delivered or are delivered more than once or in the wrong sequence. Although techniques for ensuring the reliable delivery of messages are reasonably well understood and available in some messaging middleware products today, messaging reliability is still a problem.

In most cases the SOA platform, not the service developer, is responsible for providing reliability. A common alternative is to build an SOA on top of messaging systems, such as IBM WebSphere MQ and Microsoft MSMQ.. One issue is that different products from different vendors often need bridges to interoperate.

Two specifications by the OASIS Consortium—WS-ReliableMessaging and WS-Reliability—define protocols that enable services to ensure the reliable, interoperable exchange of messages with specified delivery assurances. These specifications define four basic assurances that can be combined: *in-order delivery*, *at-least-once delivery*, *at-most-once delivery* and *exactly once delivery*.

WS-Reliability was approved as a standard in January 2005. A committee draft of WS-ReliableMessaging has been published in August 2006. It is uncertain which one will survive and be used in the future as the basis for reliable messaging. Many of the main service software vendors are now backing WS-ReliableMessaging.

5.2. Service Reliability

Service reliability means the service operates correctly and either does not fail or reports any failure to the service user. The main issue to be dealt with is managing the transactional context in order to preserve data integrity during failures and concurrent access. Transaction management is more difficult in such a distributed, loosely coupled context for two reasons. Firstly, services are usually implemented in a stand-alone fashion, and transactions begin and end within the service. Therefore, transactions that involve the composition of services require either nested transactions or a redesign of transaction demarcation. Secondly, agents performing data changes (i.e., the service providers) are distributed, and, hence, a distributed transaction model is needed. Because services may be implemented in different languages and platforms, the implementation of distributed transactions—using two-phase commit, for example—requires compatible transaction agents in all end points that interact using the same protocol.

Two different standards have been proposed that address transactions across Web services: the Business Transactions Protocol (BTP) by OASIS and the Web Services Transactions (WS-Tx) published by IBM, BEA, Microsoft, and others [10]. The architect of an SOA solution that requires transactions should understand the differences and limitations of existing standards and look for what is supported by the infrastructure to be used.

Open issues: further research is needed into how different mechanisms and approaches can help to guarantee reliability in a SOA-based system and the implications of their use on other quality attributes.

6. Availability

Availability is the proportion of time a system or component is operational and accessible when required for use. Availability of services both from the user's and provider's perspectives is a concern for the success of an SOA. From the service user's perspective, if the service is not available (even transiently), then the system cannot successfully meet its functional requirements. From the service provider's perspective, in order for the service to be used (for which the provider may receive compensation), it must be available when needed. Otherwise, the downtime could affect the provider's finances and reputation.

External service providers usually agree to provide services under a service level agreement (SLA). The SLA defines the contract for the provision of the service with details such as who provides the service, the guaranteed availability of the service, the escalation process (which is followed if the service is not handled to the service user's satisfaction), and the penalties to the provider if the service level is not met. Usually, both service provider and user offer some form of capability for monitoring the service availability and other quality-of-service requirements such as performance.

To increase availability, Web services providers can use the same techniques used for web applications in general. An infrastructure topology involving replication and load balancing is a common solution. The architect should design services to be stateless because it is much simpler to make replicas of a service available in a cluster when the service does not hold state. Furthermore, service users that rely on particular services being available must have built-in contingencies, in case the services become unavailable. For example, the application could find an alternative provider for a service.

Open issues: what contingency mechanisms are available and when is it most appropriate to use them within a system. Further research is also needed on monitoring, escalation and compensation mechanisms in SOA-based systems especially on how these can be automated within an SOA environment.

7. Modifiability

Modifiability is the ability to make changes to a system quickly and cost-effectively [8]. SOA promotes loose coupling between service users and providers. Services are self-contained, modular, and accessed via cohesive interfaces. These characteristics contribute to the creation of loosely coupled SOAs where there are few, well-known dependencies between services. Thus, the cost of modifying the implementation of services is reduced and the overall system modifiability increases. However, if service interfaces need to be changed, the change may create problems because once service interfaces are published and used by applications, it can be difficult to identify who is using a service and what impact changing its interface will have.

Extensibility, is a special case of modifiability. Extensibility is the ease with which the services' capabilities can be extended without affecting other parts of the system. It is an important quality because the business environment in which a software system lives is continually changing. These changes mean changes in the service users, service providers and the messages exchanged among them. Extending an SOA may represent:

- *Adding new services*. SOAs allow for the easy addition of services (or new versions of services) because of the dynamic binding between service users and providers, and the use of various Web standards.
- *Extend existing services without changing the interfaces*. Because services are loosely coupled, adding capabilities that do not require a change in the service interface can be done without affecting service users.
- *Extend existing services with changes to interfaces*. New capabilities that require changes to the service interface can have a broad impact on the system. In many cases, the implementation of the service users needs to change due to the interface modification. In other cases, however, additions to a given interface can be made without breaking existing service users. That is the case when SOAP extensibility mechanisms are used.

Open issues: further research is needed in processes and techniques to deal with identifying the impact of updating services and incorporating new versions of service into an SOA environment.

8. Testability

Testability is the degree to which a system or service facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [12]. Testing a system that uses an SOA is more complex for reasons that include:

- It is more difficult to setup and trace the execution of a test when the system elements reside on different machines across the network.
- The source code of external services may not be available to service users defining and running the tests. Test cases have to be defined exclusively based on the published interface and documentation. Besides, service users may not have access to log files or other outputs generated when the external service is executed.
- In some cases, services are discovered at runtime, so it may be impossible to predict which service is actually used by a system until the system executes. In addition, different services from different providers may be used at various times when the system runs. The services used may be running on different platforms or operating systems and use different middleware technologies. Building repeatable tests and automating the testing process for such a system is a challenge.
- In Web services solutions, sometimes the error is in an XML document (e.g., the WSDL interface definition). Dealing with raw XML is cumbersome, so most development tools try to protect developers from that task. But in practice Web services programmers need to understand XML very well and the structure of the documents used in the solution to solve some problems.

If a runtime problem occurs, it may be difficult to find the cause. It can be within the service user, the service provider, the communication infrastructure, the discovery agent (if one is in place), or it can be due to the load on the platform where the service executes. Trying to replicate problems in a test environment may be extremely challenging. Service providers may need to build additional services and infrastructure that support the testing and debugging processes of both the service and the service users.

Open issues: further research is needed in how to carry out testing and debugging in SOA-based systems with different platforms, and in an environment where services are discovered dynamically and it may not be known until runtime what actual services are used.

9. Usability

Usability is a measure of the quality of a user's experience in interacting with information or services. The distributed nature of SOAs can have a profound impact when the processing of a user action involves calls to remote service providers. When service calls take a long time to respond, it is usually a good idea to move the service communication to a separate thread on the application that contains the service user. In that case the user interface of that application can still be responsive

while service requests are being made. However, in the case of Web services solutions, there is no way to give the user effective feedback or control over the communications [11]. The SOAP protocol does not allow for progress notification or cancelling an active call. Progress indicator and ability to cancel a request are two features that should be available to the user when a lengthy operation is requested.

The delay introduced by the network in SOAs can be on the order of seconds. To avoid these delays, not only must the service respond to user requests with the data requested but also with other relevant data that may not be immediately displayed. Take as an example an airline reservation system where the client side GUI application is the service user. If the user enters "New York City" as the destination, the request is sent to the service provider, which can search for flights to JFK, La Guardia and Newark airports. Then results are returned to the service user application, which can show them all or ask the user to select a specific airport. In any case, the service user application already has the data, and a second call to the service provider is not necessary. Another example is an application that displays maps retrieved from a service provider. If the user is examining a particular portion of a map, common operations are panning and zooming. If the service also returns data to support panning and zooming based on the current position, it is less likely that the next user action will require invoking the map service again.

Open issues: there is a need for mechanisms to provide service users with effective feedback or control over communications. These are currently lacking in the standards. What does a usability SLA mean between a service user and provider?

10. Scalability

Scalability is the ability of an SOA to function well (without degradation of other quality attributes) when the system is changed in size or in volume in order to meet users' needs. The major issue in SOA scalability is the ability of the site where the services are located to accommodate an increasing number of service users without performance degradation.

The Web services technology does not offer any inherent scalability feature. To respond to scalability requirements, the architect has to rely on the mechanisms provided by the platform vendors. Strategies to improve scalability on the server provider side include [13]:

- *Horizontal scaling*: add load-balanced servers.
- *Vertical scaling*: increase the capacity of a server.
- *Stateless services*: design the service implementation to be stateless to avoid session management and context propagation issues.
- *Service scope*: in some platforms, one can configure whether a new instance of the service provider will be created: *(a)* for each new request, *(b)* for each new service user accessing the service, or *(c)* once for all requests from all service users. If the Web service implementation is reentrant, *(c)* is the best choice; otherwise *(a)* should be used.

If addressing scalability poses potential performance issues, the source of the delays must be identified—whether it is the transport protocol, the XML parser, the load-balancing algorithm or the SOAP runtime. The performance of the system should be studied, and load tests that capture the magnitude of the scalability (for example, if the system handles 10, 1,000, or 10,000 service users) should be performed.

Open issues: further research is needed in how scalability can be handled within SOA environments and how the uses of particular strategies affect other quality attributes.

11. Service Level Agreements and Quality Attributes in SOA Lifecycle Management

Many commercial services are being deployed using Web Services and other technologies that support the SOA architectural style. A key to success for this new business-to-business model is establishing and monitoring service level agreements (SLAs) that provide the necessary level of services to service consumers. There are still many open issues and questions to be examined further, such as: *(1)* In this new model for business-to-business relations, what mechanisms have been used to ensure quality of service by contract? *(2)* What quality properties have been and can be expressed in service-level agreements? *(3)* Should there be a separate SLA for each quality attribute? *(4)* What mechanisms have been used by service providers to achieve and monitor those qualities? *(5)* What support is there in the service technologies to express quality requirements and what's being formulated for the future in this respect?

Another open issue is defining how to deal with quality attributes and quality requirements in the full lifecycle management for SOA. Lifecycle management comprises design time (identifying and developing services), run time (defining and monitoring SLAs) and change time (upgrading and evolving services). Many aspects of the overall lifecycle management of SOA deal with quality. Many of these are still maturing and some are areas where further research and investigation needs to be carried out. For example defining processes to determine business value and translating that into quality attributes. Another example is how are all of the SLAs for an SOA-based system modeled and managed in a way that guarantees the required level of service for an entire system.

12. Conclusion

Choosing an architecture that satisfies the functional and the quality attribute requirements is vital to the success of a system. Nonetheless, the requirements that will shape the architecture design are the quality attribute requirements. SOA is becoming a popular solution to distributed systems. To create successful designs, it is important to understand how an SOA supports different quality attributes. This paper explored how an SOA impacts different quality attributes, identifying issues and tradeoffs related to them. Many of these issues have not been thoroughly researched, and many of the standards posing as remedies are immature.

If external services, or even services outside the control of the development department, are used, an SLA must be established between the parties to guarantee quality of service for essential services. Building a system that relies on third parties without the necessary agreements increases the risk of not meeting the quality attribute requirements. Failing to achieve those requirements could adversely affect an organization's ability to meet its business goals. Several open issues in service level agreements are outlined in this paper.

Determining business value and translating that to quality attributes is an area where mature processes are still lacking. Defining the overall governance for SOA lifecycle management is also an area that is still maturing. There are many open issues and research challenges in how quality attributes are dealt with in the overall lifecycle management of an SOA-based system

13. Acknowledgements

Lero is supported by Science Foundation Ireland (under grant no. 03/CE2/I303_1).

14. References

[1] Software Engineering Institute, "Software Architecture Technology Initiative", http://www.sei.cmu.edu/architecture/, 2007.

[2] L. Brownsword, et al., "Current Perspectives on Interoperability", *CMU/SEI-2004-TR-009*, Software Engineering Institute, Pittsburgh, PA, March 2004.

[3] J. McGovern, S. Tyagi, M. Stevens, and S. Matthew, *Java Web Services Architecture*, Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[4] B. Atkinson, et al., "Specification: Web Services Security (WS-Security), Version 1.0", http://www-128.ibm.com/developerworks/library/ws-secure/, April 5, 2002.

[5] OASIS, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, March 2004.

[6] Web Services-Interoperability Organization (WS-I). *Basic Security Profile 1.1 – Working Group Draft*. http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html. October 19, 2006.

[7] N. Gall and E. Perkins, "The Intersection of Web Services and Security Management: A Service-Oriented Security Architecture", http://whitepapers.ostg.com/detail/RES/1066934618_724.html, July 2003.

[8] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures*, Addison-Wesley, Boston, 2002.

[10] M. Little and T. Freund, "A Comparison of Web Services Transaction Protocols", http://www-128.ibm.com/developerworks/webservices/library/ws-comproto/, Oct, 2003.

[11] S. Loughran and E. Smith, "Rethinking the Java SOAP Stack", *2005 IEEE International Conference on Web Services (ICWS 2005)*. Orlando, Florida, July 12-15, 2005.

[12] IEEE Computer Society, *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries: 610*, Institute of Electrical and Electronics Engineers, New York, NY, 1990.

[13] O. Zimmermann, M. Tomlinson, and S. Peuser, *Perspectives on Web Services*, Springer, 2003.